Lappeenranta University of Technology
School of Engineering Science
Computational Engineering and Technical Physics
Intelligent Computing

Felix Hildén

Time Series Clustering by Extracted Features

Bachelor’s Thesis

Examiner: Professor, D.Sc (Tech) Lasse Lensu

Supervisors: Postdoctoral researcher, D.Sc. (Tech) Matylda Jablonska-Sabuka
Professor, D.Sc (Tech) Lasse Lensu
Associate professor, D.Sc (Tech) Arto Kaarna
Abstract

Lappeenranta University of Technology
School of Engineering Science
Computational Engineering and Technical Physics
Intelligent Computing

Felix Hildén

Time Series Clustering by Extracted Features

Bachelor’s Thesis

2019

44 pages, 11 figures, 3 tables, 2 appendices.

Examiner: Professor, D.Sc (Tech) Lasse Lensu

Keywords: pattern recognition, electrical consumption, clustering, self-organising map

Clustering electricity consumers can give insight into how a particular consumption profile is related to other attributes of the customer, for example the residence size or whether a sauna is in use or not. Furthermore, if the profiles are separated well enough, more accurate models predicting consumption could be built.

To cluster consumers, a set of static features were extracted from raw time series representing hourly energy consumption. These features were then clustered with a self-organising map. To visualise the results, background data detailing additional information on each consumer was used as categorical labels for each series.

The results were promising. A set of background data produced some separation on the map, which indicates that there are tangible differences to the hourly consumption data alone.
Tiivistelmä

Lappeenrannan teknillinen yliopisto
School of Engineering Science
Laskennallinen tekniikka ja teknillinen fysiikka
Älykäs laskenta

Felix Hildén

Aikasarjojen ryhmittely laskettujen piirteiden perusteella

Kandidaatin työ

2019

44 sivua, 11 kuvaa, 3 taulukkoa, 2 liitettä.

Tarkastajat: Professori, TkT Lasse Lensu

Hakusanat: hahmontunnistus, sähkönkulutus, ryhmittely, itseorganisoituva kartta

Sähkönkuluttajien ryhmittely voi kertoa, miten tietty kulutusprofiili liittyy kuluttajan muihin ominaisuuksiin, esimerkiksi asunnon kokoon ja saunan käyttöön. Jos profiilit eroavat toisistaan tarpeeksi, tarkempia kulutuksen ennusteita voitaisiin tehdä ryhmien perusteella.

Ryhmittelyn suorittamiseksi tunnittaisen kulutuksen aikasarjoista laskettiin piirteitä, jotka ryhmiteltiin itseorganisoituvan kartan avulla. Tulosten näyttämiseksi kuhunkin kulutusyhteyden liitettyjä taustatietoja käytettiin ryhmien nimikkeinä.

Tulokset olivat lupaavia. Osa taustatiedoista erottui kartalla, mikä kertoo siitä, että pelkisessä kulutuksen aikasarjoissa on todellisia eroja.
# CONTENTS

1 Introduction 6
   1.1 Background .................................................. 6
   1.2 Objectives and delimitations .............................. 6

2 Time series clustering 8
   2.1 Raw-data-based methods .................................. 8
   2.2 Feature-based methods .................................... 8
   2.3 Model-based methods ...................................... 9
   2.4 Series representation methods ............................ 9
   2.5 Conclusion .................................................. 9

3 Methods 10
   3.1 Decomposition and power transform .................... 10
   3.2 Time series features ....................................... 10
      3.2.1 Trend and seasonality ................................ 11
      3.2.2 Periodicity ............................................ 11
      3.2.3 Skewness .............................................. 11
      3.2.4 Kurtosis ................................................ 12
      3.2.5 Serial correlation .................................... 12
      3.2.6 Self-similarity ....................................... 12
      3.2.7 Non-linearity .......................................... 13
      3.2.8 Chaos .................................................. 13
   3.3 Self-organising map ........................................ 13
      3.3.1 Learning algorithm .................................... 14
      3.3.2 Learning rate .......................................... 14
      3.3.3 Neighbourhood function ............................... 15
   3.4 SOM evaluation criteria .................................. 16
      3.4.1 Projection ............................................... 16
      3.4.2 U-matrix ................................................ 16
      3.4.3 Topographic error ..................................... 16
      3.4.4 Embedding accuracy ................................... 17

4 Implementation 18
   4.1 Tools .......................................................... 18
   4.2 Algorithms .................................................. 18
      4.2.1 Periodicity .............................................. 18
      4.2.2 Optimal Box-Cox transform with decomposition .... 20
5 Experiments and results
5.1 Data
5.2 Commonalities in experiments
  5.2.1 Pre-processing
  5.2.2 Feature extraction
  5.2.3 Clustering
5.3 Experiment 1: Effect of post-feature-extraction processing
5.4 Experiment 2: Hyper-parameters of a self-organising map
5.5 Experiment 3: Clusters with labelled data

6 Discussion
6.1 Current study
  6.1.1 Shortcomings
6.2 Future work

References

Appendices
  Appendix 1: Feature distributions
  Appendix 2: Labelled data projections
1 Introduction

1.1 Background

Clustering as a type of machine learning is a technique used to extract groups of similar data from a set. Since the time of ancient Romans, clustering of people has been an integral part of society [1]. Karl Marx called these clusters classes [2] and those that belong to other groups were named class enemies [3]. Pioneered by the Russians with their Gulag prison system that encompassed hundreds of clusters across Siberia [4], the science of clustering made huge advances in the 20th century. By the mid-century, Europe was effectively clustered into two well-separated zones [5] with the exception of Berlin, using a type of adversarial algorithm.

Aghabozorgi et al. outline different types of clustering and their uses in a review of time-series-clustering-related literature [6], which is now discussed briefly. As an unsupervised learning technique, clustering is primarily used as a tool to discover patterns in data. Its objective is to devise groups of data such that intra-group differences are minimised and inter-group differences are maximised. Conventionally two types of patterns may be found. Firstly, patterns may be prominent. For example, data may form a number of clusters that hold similar data. Secondly, clustering can be used to detect outliers, where it is less important to know which cluster a sample belongs to, but detect individual samples that differ significantly from others. Recent applications use time series clustering also in recognising dynamic changes in sequences, as a basis for predictive models and in pattern discovery.

The ultimate motivation for this thesis is predicting energy consumption. By clustering consumers, per-group models can be built which may be more accurate and reliable than a model constructed for the whole data set. Predicting electricity needs is paramount to regulating supply, so the problem is not purely one of intellectual curiosity, but one of great practical utility.

1.2 Objectives and delimitations

In their 2004 article [7], Wang et al. clustered time series by first extracting static features and then using a self-organising map to group together similar series. This thesis attempts to loosely follow in its footsteps, using similar methods to cluster electricity consumption
data. Implementation details differ in feature extraction and self-organising map training, as Wang et al. have not made all information available and ready-made libraries were heavily utilised in this thesis. Additional steps were also performed particularly after feature extraction.

It would be most useful, if the resulting material of this work could be used in further similar research and even developing practical applications. Therefore, care is taken to ensure re-usability and a reasonable structure from a software engineering perspective.
2 Time series clustering

In contrast to conventional applications, clustering time series is challenging in a number of ways. Warren Liao has described these challenges and methods of overcoming them in a survey of time series clustering literature [8]. This chapter provides an overview of that discussion.

Time series are by nature high-dimensional, which can make methods utilising raw data impractical due to computational complexity. Choosing a similarity measure for clustering is not a trivial task. For example, it is not obvious whether a series and its time-shifted version should be similar or not. And if they should, how to make them alike by some measure. Noisy data can be problematic when working with raw data directly. Additionally, time series that have different lengths can intuitively be similar, but most methods do not consider feature vectors of different lengths. Not being restricted by the series length is therefore an advantage when it comes to processing time series. A number of approaches have been developed to address these issues. Most of them only consider univariate series.

2.1 Raw-data-based methods

Methods that work on raw data often aim to construct such a similarity measure that it accurately reflects what it means for two time series to be alike. For example, this might include time stretching of one or both compared sequences. Series with unequal lengths or uneven sampling intervals could be made comparable by interpolation. Though, most of the methods gathered by Liao [8] only consider series of equal length. After applying a suitable distance measure, an algorithm like c-means or an agglomerative hierarchical clustering is used to produce clusters.

2.2 Feature-based methods

Calculating static features from a time series can be beneficial. It greatly reduces computational complexity of the clustering process, particularly with long series or ones with a short sampling interval. Although the feature extraction process is often generic, extracted features need not be similar for all applications. Every feature can be chosen based on its suitability for the domain in question. Feature-based approaches better accommodate
series of unequal lengths because of the feature extraction step.

2.3 Model-based methods

Time series can be represented by some sort of underlying model, for example one that relies on moving averages or some probability distribution to represent possible model output. Series are then considered similar if the models representing individual series or the residuals of a model are similar. Like feature-based methods, using a model allows for representing series of unequal lengths.

2.4 Series representation methods

Aghabozorgi et al. have gathered additional methods of representing time series beyond raw data, extracted features or models [6]. Their aim is dimensionality reduction, resulting in reduced computational complexity in the clustering phase. Methods like discrete Fourier transform, wavelet transform and piecewise linear approximation have been used for this task.

2.5 Conclusion

Based on the article by Wang et al. [7], a feature-based method of calculating static feature vectors and using a conventional clustering algorithm was chosen as an experimental method for this thesis. Their work provides a set of universal features to describe various aspects of time series.
3 Methods

In order to reduce the complexity of time series clustering and eliminate issues with choosing a suitable similarity measure, several static features are calculated from the data instead. Thus the problem is transformed into a more conventional clustering task.

Feature extraction is considered as Wang et al. [7] have performed it unless specified otherwise. Within this chapter, features that were self-implemented are described in more detail than the ones for which an implementation was already available.

3.1 Decomposition and power transform

Time series can exhibit trend and seasonality. If values tend to increase or decrease in spite of local variability, the series has trend. Similarly, if the values tend to increase or decrease in cycles, the series is seasonal.

A decomposition can be based on simple averages [9]. First, the trend of a series over time is isolated by applying a moving average over the whole series. After subtracting trend, seasonality is estimated. Based on a fixed season length, the series is divided into periods. The mean value of the period at every index is used as the seasonal component. Finally, the season is also subtracted and only a residual component remains.

Before extracting features, the series are transformed using a Box-Cox power transform. An optimal transformation parameter $\lambda$ is chosen for each series such that $\lambda \in [-1, 1]$ and the resulting distribution of residuals after decomposition is maximally normal. The Shapiro-Wilk test was used as a normality test.

3.2 Time series features

Both the Box-Cox-transformed data and decomposition residuals are used in calculating features. First, measures of trend and seasonality are calculated based on the decomposition components. Periodicity is extracted only from the raw data. A set of features is then calculated for both raw and residual data. This set contains measures of skewness, kurtosis, serial correlation, self-similarity, non-linearity and chaos.
3.2.1 Trend and seasonality

Calculated with decomposition results, trend and seasonality attempt to measure the amount of variation explained by the respective components. For a univariate time series $y(t)$ and a corresponding de-trended series $y'(t)$, the measure of trend is

$$T = 1 - \frac{\text{Var}(y')}{\text{Var}(y)}.$$  

(1)

Similarly, for a series $y(t)$ and a corresponding de-seasonalised series $y'(t)$, the measure of seasonality is

$$S = 1 - \frac{\text{Var}(y')}{\text{Var}(y)}.$$  

(2)

3.2.2 Periodicity

*Autocorrelation* is the correlation of a time series with a delayed version of itself. When calculated using different delays it forms the autocorrelation function of the series. Strong autocorrelation at a certain offset can be a sign of periodicity in a signal, particularly if it was preceded by a deep valley. Let the period of a time series therefore be the index of a peak in the first valley-peak pair with a difference in autocorrelation above some threshold to ensure significance. If there is no such pair, periodicity is set to zero.

3.2.3 Skewness

Skewness is a measure of lopsidedness, where data which is asymmetric around its mean has a negative or positive skewness. A value of zero indicates that a distribution is completely symmetric. For a univariate time series $y(t)$ with length $n$, its skewness is

$$S = \frac{\sum_{i=1}^{n} (y(t) - \mu)^3}{n\sigma^3},$$  

(3)

where $\mu$ and $\sigma$ are the mean and standard deviation of $y$, respectively.
3.2.4 Kurtosis

Kurtosis is a measure of heavy tails. It measures high for distributions that tend to produce extreme values and low for ones that generate them near the mean. For a univariate time series $y(t)$ with length $n$, its kurtosis is

$$K = \frac{1}{n\sigma^4} \sum_{t=1}^{n} (y(t) - \mu)^4,$$

where $\mu$ and $\sigma$ are the mean and standard deviation of $y$, respectively. The kurtosis of a normal distribution is three. Let us use excess kurtosis $K_e$ as a measure of heavy tails instead. It is simply defined as $K_e = K - 3$, the kurtosis compared to that of a normal distribution.

3.2.5 Serial correlation

As a measure of serial correlation the Ljung-Box test is employed to reduce autocorrelation function (ACF) values to a single metric. The test statistic is

$$Q = n(n + 2) \sum_{k=1}^{h} \frac{\hat{\rho}_k^2}{n-k},$$

where $n$ is the series’ length, $h$ the number of ACF values that are considered and $\hat{\rho}_k$ the ACF value at delay $k$. The test statistic is used as-is, without testing the initial hypothesis of no serial correlation using the $\chi^2$ distribution.

3.2.6 Self-similarity

The Hurst exponent is a measure of the long-range dependence or persistence of values within a series. As explored by Qian et al. [10], persistence of values means that a large value is likely to be followed by other large values. Anti-persistence is indication of a mean-reverting series, where a large value tends to be followed by a small one and vice versa. An exponent of $H = 0.5$ indicates a random series. Values of $0 < H < 0.5$ are evidence of anti-persistence, and correspondingly the range $0.5 < H < 1$ is associated with persistent series. The exponent can be estimated using rescaled range analysis [10].
3.2.7 Non-linearity

Two models are constructed for a data point $y_i$ as a function of previous values. For example, a second order model would be $y_i = f(y_{i-1}, y_{i-2})$. Residuals of a linear model are compared to those of a third-degree polynomial, which includes all combinations of input variables up to three degrees. A $\chi^2$ test is then performed to determine if the data is non-linear. The test statistic for linearity is

$$s = n \cdot \log \frac{\text{SSR}_1}{\text{SSR}_3},$$

(6)

where $n$ is the series’ length and SSR$_n$ is the squared sum of the residuals of an nth-order model. Finally, the probability is converted to that of the opposite hypothesis of non-linearity.

$$p = 1 - \chi^2_d(s),$$

(7)

where $\chi^2_d$ is the $\chi^2$ probability density function with $d$ degrees of freedom. $d$ is equal to the difference in amount of terms in the non-linear model to the linear one.

3.2.8 Chaos

The Lyapunov exponents of a dynamic system describe the behaviour of nearby trajectories. Wolf et al. state that "a system with positive Lyapunov exponents is defined to be chaotic" [11]. Estimating the maximal Lyapunov exponent can be used as a measure of chaos. For a positive exponent, trajectories diverge exponentially while for a negative one they converge.

3.3 Self-organising map

Also known as Kohonen maps, self-organising maps (SOM) are now discussed on the basis of Kohonen’s 1982 article [12]. They are a class of artificial neural networks and are widely used thanks to their powerful visualisation capabilities. A self-organising map consists of a single layer of neurons with weights associated with each input feature. Unlike traditional neural networks, these weights are not used to calculate a neuron’s activation. Rather, they represent a point in the input space. In addition, the neurons themselves are arranged to a grid, usually a two-dimensional one, which is interpreted as the map to be organised.
3.3.1 Learning algorithm

Before learning, weights are initialised to small random values with either a uniform or a normal distribution. Learning is typically carried out one sample at a time. This procedure is visualised in Figure 1. First, a best matching unit is determined based on the sample’s distance to every node. The winning unit is moved closer to the sample along with a neighbourhood of other close-by nodes, determined by a neighbourhood function.

The process is repeated for each sample until learning has stopped. Over time, the effect of an update is attenuated using a learning rate function. This is done to avoid overstepping, which would mean bouncing back and forth between samples’ locations instead of converging to some fixed point. Training in mini-batches can also be considered. Using mini-batch learning, updates are not carried out one sample at a time but are instead aggregated over a number of training steps.

![Figure 1. Training of a self-organising map [13]](image)

3.3.2 Learning rate

In this thesis, an exponentially decaying function was used for the learning rate multiplier (Figure 2).

\[ L(t) = \exp \left( -\frac{4t}{e} \right), \]  

(8)

where \( L \) is a function of elapsed training epochs \( t \) and \( e \) is the estimated maximum number of training epochs.
3.3.3 Neighbourhood function

The neighbourhood function utilised a decaying Gaussian. Map size was used to scale the neighbourhood function to a suitable range.

\[ N(r, t) = \exp \left( - \left( \frac{2r}{s} \right)^2 \cdot \left( \frac{e}{e - t} \right)^3 \right), \quad (9) \]

where \( N \) is a function of radius from the best-matching unit \( r \) and elapsed training epochs \( t \). The constant parameter \( s \) is the length of the self-organising map along its longest dimension and \( e \) is the estimated maximum number of training epochs. Figure 3 shows the initial neighbourhood activation and its value as training progresses.
3.4 SOM evaluation criteria

Self-organising maps as a clustering technique do not produce clear clusters. But they are very useful for visualisation. Therefore the main evaluation criteria are also visual. A numeric way of measuring map quality as presented by Hamel [14] is used to provide immediate information on the result’s quality. Hamel suggested determining the embedding accuracy and topographic error of a map to produce a reliable but computationally efficient way of evaluating a map.

3.4.1 Projection

Each time series can be projected onto the map by background data labels. One can then tell at a glance whether each class is separated on the map. This can also be done without labels to provide some understanding of the map shape.

A simple way of assigning clusters to the map is by the majority vote of data points that land at each node. To extend this approach, percentages can be used to represent the purity of a node. Multiple labels can be associated with each node, as the background data has multiple dimensions.

3.4.2 U-matrix

A unified distance matrix is used to visualise map topology. The mean Euclidean distance of each node to its neighbours within some range is rendered as a greyscale image. Thus, clusters of nodes can be identified as a group of close-by units that are surrounded by a border. In this implementation the colours are inverted. White represents nodes that are close to each other, black those that are far away.

3.4.3 Topographic error

To measure the topological quality of a map, best-matching units are paired with second-best-matching units and their adjacency on the map is determined. A simple measure of topographic error [14] is then

\[ T_e = \frac{1}{n} \sum_{i=1}^{n} \text{err}(x_i), \]  
(10)
where $n$ is the number of samples and

$$\text{err}(x_i) = \begin{cases} 
1, & \text{if the best and second-best-matching units} \\
& \text{of sample } x_i \text{ are not neighbours} \\
0, & \text{otherwise}. 
\end{cases}$$

(11)

### 3.4.4 Embedding accuracy

To test whether a map has embedded each feature, the distributions of nodes and data samples are compared. Hamel used an F-distribution to test the equality in both mean and variance [14]. In this thesis, a two-sided Kolmogorov-Smirnov test provided with the self-organising map implementation is used instead. A feature is considered as embedded if the test is passed. The total embedding accuracy of a map is

$$E = \frac{1}{d} \sum_{i=1}^{d} \rho_i,$$

(12)

where $d$ is the number of feature dimensions and

$$\rho_i = \begin{cases} 
1, & \text{if feature } i \text{ is embedded} \\
0, & \text{otherwise}. 
\end{cases}$$

(13)
4 Implementation

4.1 Tools

Python has solidified its place in data science and machine learning largely thanks to the NumPy package. By introducing an efficient n-dimensional array implementation and vectorised operations it allowed heavy computations to be orchestrated with an otherwise unsuitable but expressive language. Higher-level packages that build on top of NumPy make data analysis tasks effortless. Packages that were used are described in Table 1. There were no external requirements placed on tools or algorithms. In general, if a ready-made solution was available, it was used to reduce the time spent on development.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h5py</td>
<td>HDF5 file operations, used for opening Matlab .mat-files</td>
</tr>
<tr>
<td>nolds</td>
<td>Non-linear dynamic systems, used in feature extraction</td>
</tr>
<tr>
<td>numpy</td>
<td>N-dimensional typed array and vectorised operations</td>
</tr>
<tr>
<td>pandas</td>
<td>Database-style table operations</td>
</tr>
<tr>
<td>scipy</td>
<td>Scientific computing, used in feature extraction</td>
</tr>
<tr>
<td>sklearn</td>
<td>Machine learning, used in post-processing</td>
</tr>
<tr>
<td>statsmodels</td>
<td>Statistics, used in feature extraction</td>
</tr>
</tbody>
</table>

4.2 Algorithms

4.2.1 Periodicity

To find the periodicity of a series, a procedure described in Algorithm 1 was used. The autocorrelation function of the series was calculated for lags up to a third of the series’ length. By subtracting adjacent values from one another, change in autocorrelation was used to find peaks and valleys of the ACF. Peaks are found where the sign changes from positive to negative and vice versa for valleys. These indices were then scanned for a valley-peak pair that had no other valley or peak in between and were associated with a difference in autocorrelation of over 0.1.
Algorithm 1 Periodicity

Require: autocorrelation function $A(i)$
Require: threshold $t$

1: $d(i) := A(i + 1) - A(i)$  \hspace{1em} \triangleright \text{pairwise differences}$
2: $s(i) := \text{sign}(d(i))$
3: $e := e(i)$  \hspace{1em} \triangleright \text{extrema of } A \text{ to be calculated}$

4: for all $i$ do
5: \hspace{1em} if $s(i) = s(i + 1)$ then
6: \hspace{2em} next \hspace{1em} \triangleright \text{consider extrema only}$
7: \hspace{1em} end if
8: \hspace{1em} if $d(i) < 0$ then
9: \hspace{2em} $e(i) := -1$  \hspace{1em} \triangleright \text{minimum}$
10: \hspace{1em} else if $d(i) > 0$ then
11: \hspace{2em} $e(i) := 1$  \hspace{1em} \triangleright \text{maximum}$
12: \hspace{2em} else
13: \hspace{3em} $e(i) := 0$
14: \hspace{1em} end if
15: end for
16: for all extrema do
17: \hspace{1em} if current extremum is a maximum, previous extremum was a minimum and the associated difference in $A$ is above $t$ then
18: \hspace{2em} return periodicity as the index of the current extremum
19: \hspace{1em} end if
20: end for
21: no periodicity was found, return 0
4.2.2 Optimal Box-Cox transform with decomposition

The criteria of optimality in the Box-Cox power transform used on each series was not normality of the data itself, but of the residuals after decomposition. For this reason, the optimiser (detailed in Algorithm 2) had to be written from scratch. A simple narrowing grid search was employed. First, a linear range of values between $-1$ and 1 were chosen as candidates. The series was transformed using each $\lambda$ candidate. After decomposition, normality of the residuals was tested using the Shapiro-Wilk test. A new range was constructed between the adjacent values of the optimum. This procedure was then repeated.

Ranges of ten points were used for each iteration, and three iterations were performed to find a suitably accurate optimum. If the optimal value was ever exactly $-1$ or 1, it was shifted towards zero to keep the search strictly within the set borders.

**Algorithm 2** Box-Cox transformation with decomposition

```
Require: s ⊿ series as array
Require: n ⊿ number of search passes
1: if min(s) < 0 then ▷ Box Cox transformation requires strictly positive data
2:    return with no transformation
3: else if min(s) = 0 then
4:    s := s + 0.001 \cdot max(s)
5: end if
6: α := 0 ▷ current optimum
7: for i in range(0, n − 1) do ▷ range(start, stop)
8:    if α = −1 then
9:        α := α + 10−i
10:   else if α = 1 then
11:        α := α − 10−i
12: end if
13: for λ in range(α − 10−i, α + 10−i, 10−i−1) do ▷ range(start, stop, step)
14:    t := box_cox(s, λ)
15:    r := decompose_residuals(t)
16:    p(i) := shapiro_wilk(r) ▷ gather all test results
17: end for
18: α := λ that produced optimal p
19: end for
20: return α
```
4.3 Ready-made implementations

Implementations found in open-source Python libraries mentioned in Table 1 were heavily used in feature extraction and post-processing. They were used as detailed in Table 2.

<table>
<thead>
<tr>
<th>Package</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scipy</td>
<td>boxcox</td>
<td>Box-Cox power transform for raw data</td>
</tr>
<tr>
<td>statsmodels</td>
<td>acf</td>
<td>Autocorrelation function for periodicity</td>
</tr>
<tr>
<td>statsmodels</td>
<td>seasonal_decompose</td>
<td>Seasonality and trend decomposition</td>
</tr>
<tr>
<td>statsmodels</td>
<td>acorr_ljungbox</td>
<td>Serial correlation</td>
</tr>
<tr>
<td>nolds</td>
<td>hurst_rs</td>
<td>Hurst exponent</td>
</tr>
<tr>
<td>nolds</td>
<td>lyap_r</td>
<td>Chaos as the maximal Lyapunov exponent</td>
</tr>
<tr>
<td>sklearn</td>
<td>PowerTransformer</td>
<td>Yeo-Johnson transform</td>
</tr>
<tr>
<td>sklearn</td>
<td>RobustScaler</td>
<td>Quartile range scaling</td>
</tr>
</tbody>
</table>

An implementation of the self-organising map was available from the author's previous work. In addition to the algorithm itself, the package includes visualisations and measures that were used as mentioned in Section 3.
5 Experiments and results

5.1 Data

An electrical power company provided hourly consumption data of some 3400 locations recorded for one year. Additional information was also available detailing for example the type of heating system, ventilation and whether a sauna was in use. This background data, however, was only available for 900 locations. The data was not gathered specifically for this work. It was already available after related research efforts and provided as-is.

5.2 Commonalities in experiments

5.2.1 Pre-processing

Before extracting features, the series were pre-processed. Missing values in the data were represented as not-a-number (NaN) values. Some series were completely devoid of valid data points. They were discarded along with redundant, identical duplicate entries. Some series contained less frequent missing values. Linear interpolation was used to fill them in, both forward and back.

5.2.2 Feature extraction

Trend and seasonality decomposition was carried out on each series. Measures of trend and seasonality were calculated based on the decomposition. Periodicity was determined for the raw data. Finally, measures of skewness, kurtosis, serial correlation, self-similarity, non-linearity and chaos were calculated for raw data and the residual portion of the decomposition. As most series were almost certainly non-linear, the measure was converted to one of linearity: $L = 1 - N$.

5.2.3 Clustering

The data were clustered using a two-dimensional self-organising map. Small dimensions of maps were preferred to enable clear visualisations and faster computation. Weights
were initialised using a normal distribution and training was done one series at a time.

5.3 Experiment 1: Effect of post-feature-extraction processing

The feature distributions were far from normal and had many outliers. This lead to difficulties with weight initialisation not adequately representing the data and ultimately to poor map convergence.

To aid in the clustering process, feature distributions were normalised using a Yeo-Johnson power transform, which is conveniently also defined for negative data, unlike the Box-Cox transform. Figure 4 depicts example distributions for kurtosis and linearity of raw data, both before and after Yeo-Johnson transform. The distributions are visualised first in their un-processed state and then after a Yeo-Johnson transform along with a reference normal distribution. All feature distributions can be found in Appendix 1. As can be seen from the measure of kurtosis and other features, most distributions were normalised well, but particularly measures of linearity did not produce a normal distribution.

Results of the clustering process varied wildly depending on whether post-processing was applied. Figures 5 and 6 show that very difference. The U-matrix shows that without post-processing there are a group of outliers in the upper-left region of the map. Most samples are located in the lower-right region, as visualised by the map projection where bright colours indicate a large amount of data associated with a node. In addition to the clear visual distinction, without post-processing the topographic error was significantly higher and no feature was embedded.

Whether simple feature scaling as seen in Figure 7 or a Yeo-Johnson transform was applied (Fig. 6), made little difference in visual map quality. However, the embedding accuracy of a map trained with scaled data was lower than of one with power-transformed data. Based on these results, features were Yeo-Johnson transformed in later experiments.

5.4 Experiment 2: Hyper-parameters of a self-organising map

Using different hyper-parameters might yield better results. For a self-organising map, one can adjust the initial weights and learning rate, the learning rate function, neighbourhood function, map dimensions, their length, the number of training epochs and the size of a mini-batch.
Figure 4. Feature distributions before and after Yeo-Johnson transform

Figure 5. U-matrix and map projection without post-processing
Figure 6. U-matrix and map projection with Yeo-Johnson transform

Figure 7. U-matrix and map projection with feature scaling
Table 3. Effect of different hyper-parameters

<table>
<thead>
<tr>
<th>Size</th>
<th>Epochs</th>
<th>Learning rate</th>
<th>Weights</th>
<th>Topographic error</th>
<th>Embedding accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10000</td>
<td>0.1</td>
<td>normal</td>
<td>0.465</td>
<td>0.667</td>
</tr>
<tr>
<td>20</td>
<td>1000</td>
<td>0.1</td>
<td>normal</td>
<td>0.461</td>
<td>0.467</td>
</tr>
<tr>
<td>20</td>
<td>1000</td>
<td>1.0</td>
<td>normal</td>
<td>0.503</td>
<td>0.600</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>0.5</td>
<td>normal</td>
<td>0.520</td>
<td>0.600</td>
</tr>
<tr>
<td>40</td>
<td>1000</td>
<td>0.5</td>
<td>normal</td>
<td><strong>0.424</strong></td>
<td>0.400</td>
</tr>
<tr>
<td>20</td>
<td>1000</td>
<td>0.5</td>
<td>uniform</td>
<td>0.489</td>
<td>0.600</td>
</tr>
<tr>
<td>20</td>
<td>1000</td>
<td>0.5</td>
<td>normal</td>
<td>0.489</td>
<td>0.600</td>
</tr>
<tr>
<td>40</td>
<td>10000</td>
<td>0.5</td>
<td>normal</td>
<td>0.448</td>
<td>0.533</td>
</tr>
</tbody>
</table>

Table 3 shows parameters and their associated topographic error and embedding accuracy after training. *Size* refers to the number of nodes along both map dimensions. The largest embedding accuracy was achieved with a long training time and the lowest topographic error with the largest map. Interestingly, when combined a long training time and a large map did not produce better results. The U-matrices associated with each map were similar, as seen in Figure 8. Due to random weight initialisation they could rotate about any axis, but shared a structure even across map sizes. Thus, the effect of hyper-parameter selection is quite small. To provide clear visualisations, the smallest of map sizes are used in later experiments.

### 5.5 Experiment 3: Clusters with labelled data

Background data associated with each series was used to assign them to multiple classes. Their distribution when projected onto a map could give insight to the clustering result and the importance of each piece of background data.

The data points were projected onto the map and grouped by each category of background data. The percentage of each class at a node was visualised with pie charts. They can be found in Appendix 2. Particularly labels indicating the residence type and size, type of ventilation, whether a sauna is in use and to what degree it is used were found to be visually separate on the map.

Further visualisation was employed to assess the interesting labels. Label maps were produced to visualise the most frequent label at each node. However, the classes were often imbalanced, so label maps were misleading. Each class of every label was therefore visualised in a separate heat map. As displayed in Figure 9, the type of residence greatly affects the map position associated with each series. A similar distinction can
Figure 8. U-matrices with different hyper-parameters

(a) Uniform initialization
(b) High learning rate
(c) Large map
(d) Small map
be made with the ventilation type in Figure 10. Finally, Figure 11 visualises heat maps for three categories: no sauna in use, sauna in use, and the category associated with the highest amount of sauna usage. It seems that there are two main clusters on the map. One associated with a lower energy consumption profile and another with a higher one.
Figure 11. Sauna usage
6 Discussion

6.1 Current study

The main objective of this thesis, clustering time series data by calculating static features and utilising a self-organising map, was met. Experimental results were satisfactory. The produced maps were able to separate different series from one another, as seen in the experiment utilising background data. Although the separation was far from clear, it indicates that there are tangible differences in the consumption data alone. A library of code able to reproduce these results was constructed and refined to benefit future experiments.

6.1.1 Shortcomings

Some less-than-ideal, arbitrary and outright questionable decisions were made during the experimental portion of this thesis. Most of them were related to feature extraction and the self-organising map algorithm.

1. Calculation of periodicity. As suggested by Wang et al. [7], the threshold for periodicity’s valley-peak difference was set to 0.1 without further validation. It is also unclear, what is the effect of using an unbiased autocorrelation function instead of the biased one that was used.

2. Calculation of serial correlation. The number of lags to use in the Ljung-Box test was set to 1000. This was based on plotting the autocorrelation functions and choosing an index suitably far, where the ACF had settled to one value.

3. Calculation of chaos. The minimum separation in Lyapunov exponents’ computation was arbitrarily set to one week (24 · 7 = 168 data points).

4. Optimising the Box-Cox transformation. A better search algorithm, or a more fine-grained space of points and more iterations could be used to find a better optimum.

5. Learning rate and neighbourhood. The functions were constructed with pre-conceived notions of what they should look like. No validation or comparisons were made to other functions.
6.2 Future work

The work in this thesis is by no means conclusive or comprehensive. In addition to the aforementioned shortcomings, further work should be aimed at three particular levels of analysis.

1. Optimising the current algorithm. Although some effort was put into choosing suitable hyper-parameters, no systematic search was conducted.

2. Determining the importance of each feature and introducing new features. The measures that were used are generic descriptors of time series. Whether those features were suitable for analysis of electricity consumption data was not considered.

   For example, the measure of serial correlation tended to first rise with some slope and then plateau after some value. These characteristics could give more detailed information on the series’ behaviour.

3. Considering the suitability of the overall approach. Since the ultimate motivation of this work is to predict electricity consumption, perhaps explicit clustering of consumers is not the optimal way of achieving such a model. More modern methods even involving deep learning might be appropriate.

Other research questions arise, too.

1. Are the categories and features persistent? Do consumers shift over time?

2. Do these categories relate to other groupings that have been produced?
REFERENCES


Appendix 1. Feature distributions

![Histograms for feature distributions](image1)

Measure: raw_trend

Measure: raw_seasonality

Measure: raw_periodicity

(continues)
Appendix 1. (continued)
Appendix 1. (continued)
Appendix 1. (continued)
Appendix 1. (continued)
Appendix 2. Labelled data projections

Class pies: type

Class pies: residents

(continues)
Appendix 2. (continued)
Appendix 2. (continued)

Class pies: wood consumption

Class pies: heating type
Appendix 2. (continued)

Class pies: ventilation

Class pies: water heating

(continues)
Appendix 2. (continued)